U N I V E R S I T Y O F T A R T U
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
Institute of Computer Science
Information Technology specialty

Madis Nõmme

Handling resource intensive hybrid cloud services from iOS devices

Bachelor Thesis (6 EAP)

Supervisor: Satish Srirama, PhD

Author: ..... "....." June 2011
Supervisor: ..... "....." June 2011
Professor: ..... "....." June 2011

TARTU 2011

Contents

List of Figures

Abstract

Cloud computing and mobile computing domains have advanced rapidly and are promising technologies for the future. As the number of mobile devices continues to grow the applications for them also become more sophisticated. To satisfy the need for processing large amounts of data, developers can benefit from using various cloud resources. Cloud computing becomes mobile when a mobile device tries to access the shared pool of computing resources provided by the cloud, on demand. Mobile applications take advantage of the elasticity of the cloud to remotely process the process intensive tasks at the same time maintaining a soft real time response behavior. To support accessing the cloud services from the mobile phones, Mobile Cloud Middleware (MCM) has been realized by Srirama et al. at UT. MCM mainly handles the interoperability issues for services accessed from multiple clouds. However, when a mobile application needs to perform a task which is expected to be time consuming, as it requires involving intensive data processing and depends on mobile cloud services, it cannot hang the mobile phone until the response arrives. As a solution to address these issues, the thesis incorporated asynchronous notification framework into the MCM, especially for iOS devices.

1 Introduction

The use of smart phones and tablets has grown rapidly in recent years. Two of the main platforms - iOS and Android both have central markets for purchasing applications. As of January 22, 2011 Apple's App Store had over 350,000 applications available. It can be said that most of the "simple" applications have been already made. To be successful, developers need to create more sophisticated and more complex applications. One way to creating something new and exciting would be to make use of large amounts of data from various sources. Processing of this data while retaining relatively fast response times presents some challenges.

Mobile devices today, despite being quite powerful, have constraints on computation power, storage, network bandwidth and battery capacity. This can be limiting to application developers. One example would be a face recognition application that compares a picture against 10,000 others found from a social networking site (e.g. Facebook). If one picture averages 500kB of size then the application would need to download about 5GB of data.

Another problem arises when large amounts of data need to be exchanged between different cloud providers. When a mobile device is central piece of those transactions, all the data needs to go through it. Taking into account the bandwidth constraints of a mobile device where the best case would be a WiFi network with 54Mbps bandwidth, this solution is sufficient for only small amounts of data.

An architecture called Mobile Cloud Middleware(MCM)[13] realized by Srirama et al. at UT, is described in this thesis. MCM offers an easy way for developers to implement the kind of behaviour where resource intensive tasks are implemented outside the mobile phone (e.g. in server or cloud). When these tasks are completed, results need to be communicated back. This thesis investigates this scenario and offers a solution using asynchronous notifications.

The thesis is divided into eight chapters. First chapter describes current mobile platforms and cloud services available. Second chapter explains the idea of Mobile Cloud Middleware and some of it's architectural aspects. Third chapter talks about what is mobile cloud computing, when could it be used and the issues that come with it. Fourth chapter addresses asynchronous messaging capabilities of iOS devices and how it is used in MCM. Fifth chapter gives an overview of the prototype applications developed during the writing of this thesis. Sixth chapter explains performance analysis of MCM. A synchronization method to make timestamps taken from different systems running separate clocks comparable, is described. Results of performance analysis are also provided. Seventh chapter concludes the topics discussed. Eighth chapter offers some ideas about how the ideas presented could be further developed and tries to uncover to some of the issues that current implementation has.

2 State of the art

In the time being there are multiple cloud resource providers on the market. They include Amazon Web Services(S3, EC2, SimpleStorage)[2], Rackspace [3] and others. Most of them also provide APIs for mobile devices. A tendency among mobile application developers has been seen where the new applications make use of cloud resources. A recent study released by ABI Research says that limited processing power, battery life and data storage will limit mobile application growth in the mass market, even among smart phones like Apple's iPhone. Applications that connect to cloud resources are much likely to be successful than those that run only on the mobile device.

ABI Research[1] predicts mobile cloud computing will deliver annual revenues topping $20 billion over the next five years. ABI Research senior analyst Mark Beccue says device fragmentation and memory currently limit the level of sophistication developers can deliver through mobile applications. By contrast, sunning mobile applications in the cloud will free up mobile processors while also enabling developers to create just one version of their application. [4, 11]

In April 2011 the top 50 paid applications in Apple's App Store were mostly entertainment oriented. Although they use data sharing capability provided by Game Kit it can not be considered as using cloud resources. Interestingly among free applications the number of applications that used cloud in some way was considerably bigger. Some examples would be Skype, DropBox, Find My iPhone, Facebook.

Although it is acknowledged that using cloud resources can bring benefits to application (cost, scalability, interoperability) not many applications seem to use them. One of the reasons could be that developers don't have good procedures and architecture for quickly setting up reliable communication protocols for their mashup applications. Development periods for many of the mobile applications are quite short(in range of 3-6 weeks). If developers would have to spend 20% of that time on figuring out how to consume resources from cloud, it often is not reasonable and the idea is abandoned. By having proven procedures and basic architecture new project could be set up with basic communication working in less than 2 hours.

2.1 Cloud layers

Cloud services are provided on demand and at different levels. Shown on 2.1 on the following page are the layers of cloud services, in terms of level of abstraction. The provisioning of services can be at the Infrastructural level (IaaS) or Platform level (PaaS) or at the Software level (SaaS). In the IaaS, commodity computers, distributed across Internet, are used to perform parallel processing, distributed storage, indexing and min-

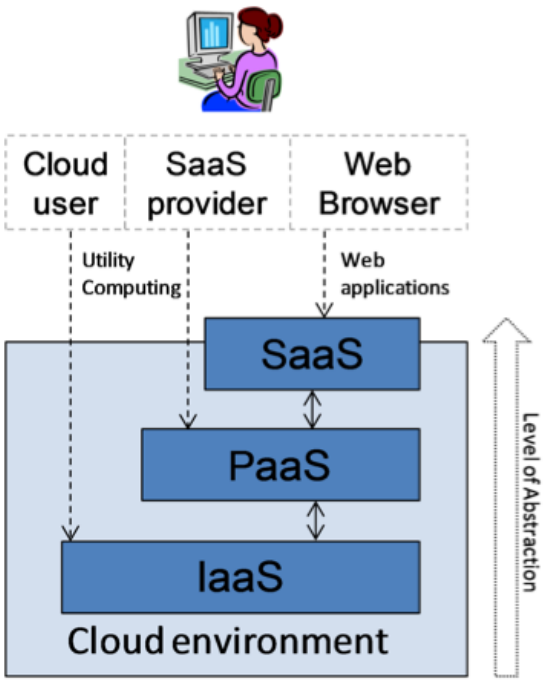

Figure 2.1: Level of abstraction and layers of cloud services [12]

ing of data. IaaS provides complete control over the operating system and the clients benefit from the computing resources like processing power and storage, e.g. Amazon EC2. Virtualization is the key technology behind realization of these services. PaaS mainly provides hosting environments for other applications. Clients can deploy the domain specific applications on these platforms, e.g. Google App Engine. These applications are in turn provided to the users as SaaS. SaaS are generally accessible from web browsers, e.g. Facebook. Web 2.0 is the main technology behind the realization of SaaS. However, the abstraction between the layers is not concrete and several of the examples can be argued for other layers.

2.2 Candidate mobile platforms

There very wide variety of different mobile phones and other mobile devices in use. Not all of them are suitable for using cloud services. Two groups of devices that are most likely targets of mobile developers wanting to use cloud services are mobile phones with iOS or Android operating systems. Both of them have mature application distributing infrastructure, large user and developer base. Also the devices have reasonably recent hardware which gives developers more freedom. In this thesis iOS platform was targeted.

```
- (void)exampleS3Usage {
    AmazonS3Client *s3 = [[AmazonS3Client alloc] initWithAccessKey:ACCESS_KEY
withSecretKey:SECRET_KEY];
    NSArray *bucketList = [s3 listBuckets];
    for (S3Bucket *bucket in bucketList) {
        // Do something with a bucket
    }
}
```

Figure 2.2: Using Amazon S3 buckets directly from iOS device

2.3 iOS platform and devices

iOS is mobile operating system developed by Apple. Most recent version of iOS is 4.3.3. iOS is used by following Apple devices: iPhone, iPod, iPad and Apple TV. The operating system consists of four layers: Core OS, Core Services, Media, Cocoa Touch.

Applications for iOS are usually written using Objective-C language which is object-oriented programming language that adds Smalltalk-style messaging to the C programming language. The iOS API called CocoaTouch is also written in Objective-C. Although regular C and C++ can be used it is not often done.

Development for iOS devices is usually done using Xcode IDE and it's building system. The building system can be invoked from IDE or from command line using xcodebuild command. iOS applications can be run on simulator or device. There are no restrictions for running them on simulator. To run application on device being member in the Apple Developer Program. Basic membership costs 99$ per year.

2.4 Using cloud services directly from iOS device

To measure application binary differences author compiled the example provided by Amazon. It demonstrates the use of Simple Storage Service (S3, `aws.amazon.com/s3`), SimpleDB (`aws.amazon.com/simpledb`), Simple Queue Service (SQS, `aws.amazon.com/sns/`), Simple Notification Service (SNS, `aws.amazon.com/sns/`). For this application Amazon Web Services SDK `http://aws.amazon.com/sdkforios/` (AWS SDK) needs to be added to be added into the project. The SDK version 0.2.3 (as of May, 25th) is 53MB. A simple application compiled and linked against the AWS SDK framework is 23MB. That is quite big considering that an template application's binary is about 25kB.

Accessing Amazon cloud services from mobile using AWS SDK is very straight forward. Entities as well as operations are encapsulated into Objective-C objects. All requests to different services are done through their according client class (AmazonS3Client, AmazonSimpleDBClient, AmazonSQSClient, AmazonSNSClient). Requests to the services are synchronous so they should not be done in main thread because application could be killed if main thread is blocked for too long.

When use of different services is required the size of application would increase even more.

3 Mobile cloud services

Cloud computing becomes mobile when a mobile device tries to access the shared pool of computing resources provided by the cloud on demand.

Mobile cloud services use the shared pool of computing resources provided by the clouds to get the process and storage intensive tasks done, from the resource constrained smart phones. Some of the well known mobile cloud services are the services provided by the social network sites like the facebook mobile, twitter mobile etc. There are other services that help in building collaboration and data sharing apps like Google docs and zoho suite, which are also available to mobile users. Mobile cloud services like Picasa and flickr are also available for sharing photos and videos. Most of the other well known public cloud services like Google maps and Amazon S3 are also accessible from mobiles.

Problems with using cloud resources arise when an application needs to access resources from different providers. Example of that would be if an application would require transferring of data from one cloud to another (for example for processing). This would have to be done by pulling the data to device.Usually the API-s are not interoperable. Thus additional processing would need to be done in device to get the data in proper format for the target cloud resource.

Other issue with using cloud services directly from device comes from application binary being tied to cloud providers implementation for accessing the resources. When application developers find a more efficient way of carrying out a task or decide to use different cloud provider all application users would need to update their application.

When application gets it's resources through service that adds an extra layer of abstraction between cloud resources and device, developers are free to change the choice of actual resources provider.

At the core level, the main services provided by cloud infrastructure are, generally, the storage service and the processing service. These services are basic in the creation of composite services which are delivered as SaaS, for instance Picasa from Google. A cloud application relies on the basic services to achieve the inherent data intensive computation performance. Accessing basic services from different cloud providers implies new types of applications, in which data saved on one storage service can be processed using a different cloud processing service, given as a result, a truly mashup application.

4 Mobile Cloud Middleware

While several mobile cloud services are existing today, most of them are bounded by numerous constraints like the mobile platform restrictions, cloud providers' technology choices etc. They provide proprietary APIs and routines to consume the services. Therefore cloud interoperability is often not possible and when a lighter mobile application needs to be created, it has to be developed for a specific cloud provider. For example: an application created on Android using jets3t to access S3 from Amazon, and Walrus from Eucalyptus, has to suffer some changes in it's configuration when it tries to access each of them. Even though Eucalyptus is compatible with Amazon infrastructure, there is no full integration between them.

Mobile Cloud Middleware (MCM) is a set of server applications that enable developer to use cloud services from different providers. MCM itself can be run in the cloud as for example in Amazon EC2. Server-side applications that were written for this thesis were developed using Java Servlet API. Apache Tomcat was used as a servlet container. Apache Maven was used for build automation and project management. Information exchanged between server and mobile device was JSON (JavaScript Object Notation) encoded.

To give the application user feedback about the tasks being carried out in MCM, messaging mechanism is integral part of MCM. Benefits of this architecture are that the server application has larger storage and bandwidth. When implementing a new feature on MCM, developer will have all the benefits with very little overhead. Adding a feature in MCM basically means to implementing one method (performTask()) on MCM.

4.1 Architecture of MCM

MCM is introduced as an intermediary between the mobile phones and the cloud. The architecture is shown in 4.1 on the next page. MCM fosters mobile platform heterogeneity and the combination of different cloud services into a mobile mashup application. When an application tries to connect to a basic cloud service, it connects to the TP Handler component of the middleware, which receives the request. The transportation handler can receive the requests based on several protocols like the Hypertext Transfer Protocol (HTTP) or the Extensible Messaging and Presence Protocol (XMPP). The request is then processed by the Interoperability API engine, which selects the suitable cloud API and creates an unique adapter that ensures the transactional process with the cloud. When the request is forwarded to the MCM Manager, it first creates a session assigning a unique identifier for saving the system configuration of the handset (OS, clouds credentials, etc.) and the service configuration requested (list of services, cloud

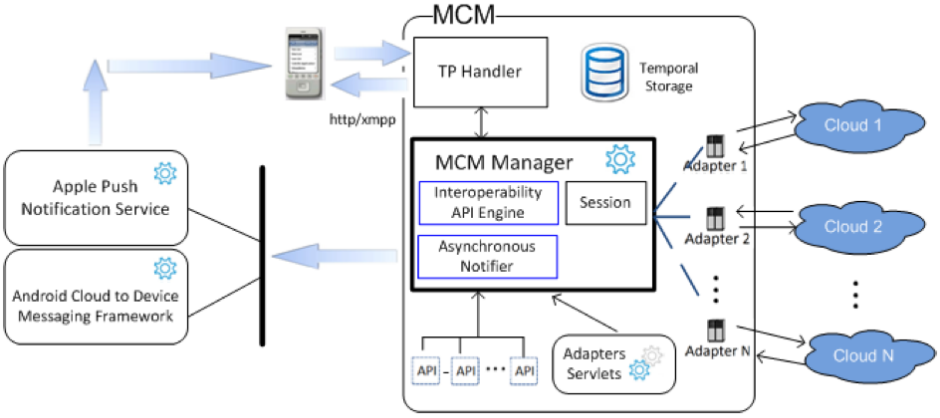

Figure 4.1: General architecture of the Mobile Cloud Middleware

providers, types of transactions, etc.) in a temporal storage space, respectively. The identifier is used for handling different requests from multiple mobile devices and for sending the notification back when the process running in the cloud is finished. Later, the interoperability API engine verifies the service configuration for selecting the suitable API, depending on the cloud vendor. A temporal transaction space is created for exchanging data between the clouds. The aim of the temporal space is to avoid offloading the same information from the mobile, again and again. Once the interoperability API engine decides which API set it is going to use, the MCM Manager requests for the specific routines from the Adapter Servlets. The servlets contain the set of functions for the consumption of the cloud services. Finally, MCM Manager encapsulates the API and the routine in an adapter for performing the transactions and accessing the SaaS. The result of each cloud transaction is sent back to the handset in a JSON (JavaScript Object Notation) format, based on the application design.[6]

4.2 Achieving asynchronous behaviour

When performing time consuming tasks on mobile asynchronous behaviour is necessary. The reasoning behind is that when the remote task is completed device can not be counted to be reachable. There are number of occasions where this might happen: user might go out of radio coverage or WiFi range. The battery of a device might run empty or user could close the application. Less probable but still possible is that user removes the application before receiving the results. All those need to be taken into account.

There are two sides of asynchronous behaviour: asynchronocity when making the request and asynchronocity during the whole time of processing requested task.

```
- (NSOperation *)taskWithData:(id)data {
    NSInvocationOperation *operation = [[NSInvocationOperation alloc]
                                        initWithTarget:self
                                        selector:@selector(doRealWork:)
                                        object:data];
    return [operation autorelease];
}

- (void)doRealWork:(id)data
{
    // Real work being done here
}
```

Figure 4.2: Using NSInvocationOperation to achieve asyncronous behaviour

```
- (void)blockOperationExampleWithData:(id)data
{
    void (^ExampleTaskBlock)(void) = ^(void){
        // Do real work
    };
    NSBlockOperation *blockOperation = [NSBlockOperation
blockOperationWithBlock:ExampleTaskBlock];
    NSOperationQueue *queue = [[NSOperationQueue alloc] init];
    [queue addOperation:blockOperation];

}
```

Figure 4.3: Using NSBlockOperation for asynchronous behaviour

4.2.1 Asynchronocity while making the request

In general sense achieving this kind of asynchronocity means making the actual request in other thread than where it was requested. In iOS devices with Cocoa based applications there are three main options available: using NSInvocationOperation object with object and selector, creating operation block and using NSBlockOperation with queue, subclassing NSOperation.

Examples of these options are provided in 4.2, 4.3 and 4.4 on the following page.

4.2.2 Asynchronocity during processing of the request

Asynchronocity during processing of the request means that results of the requested task would have to be available after the task is completed. Request initiator would have to be notified when results are ready and also provided information about where to go looking for them. Since the cloud API-s currently do not provide such functionality, this could be done by developing web service that takes in the requests and input data. It stores information about each request and when requested task is completed, informs the initiator about particular task being complete. Initiator can then request details of the results and finally go and get the results.

```
@interface Task2 : NSOperation {
}
@end

@implementation Task2
- (void)main
{
    @try {
        NSAutoreleasePool* pool = [[NSAutoreleasePool alloc] init];

        // Do some work on myData and report the results.
        [pool release];
    }
    @catch(...) {
        // Do not rethrow exceptions.
    }
}
@end
```

Figure 4.4: Subclassing NSOperation

```
public void notifyOwner() {
    NotificationCentre.getProviderForDevice(getOwnerDevice())
    .withDeviceData(getDeviceID())
    .withMessage(getMessage())
    .withKeyValuePairs("taskID", taskID)
    .send();
}
```

Figure 4.5: Using notification provider

4.3 Messaging mechanism in MCM

Asynchronous messaging mechanism was implemented in the server code. Main goals designing it were that it would have to be able to handle sending push notifications to different types of devices in similar fashion and be extensible when adding new devices in future.

Sending messages to iOS devices using Apple Push Notification Services was implemented using notnoop's Java library [5]. This project's Maven repository is available at `com.notnoop.apns`. Messaging Android devices was done using Android Cloud to Device Messaging Framework(`http://code.google.com/android/c2dm/`).

When some object in MCM needs to send out push notification, it will use NotificationCentre to instantiate needed NotificationProvider implementation. Example is given in 4.5.

4.4 Asynchronous messaging for iOS devices in MCM

Different approaches of giving feedback were considered when developing the MCM. It had to be taken into consideration that tasks being run on MCM can take undetermined

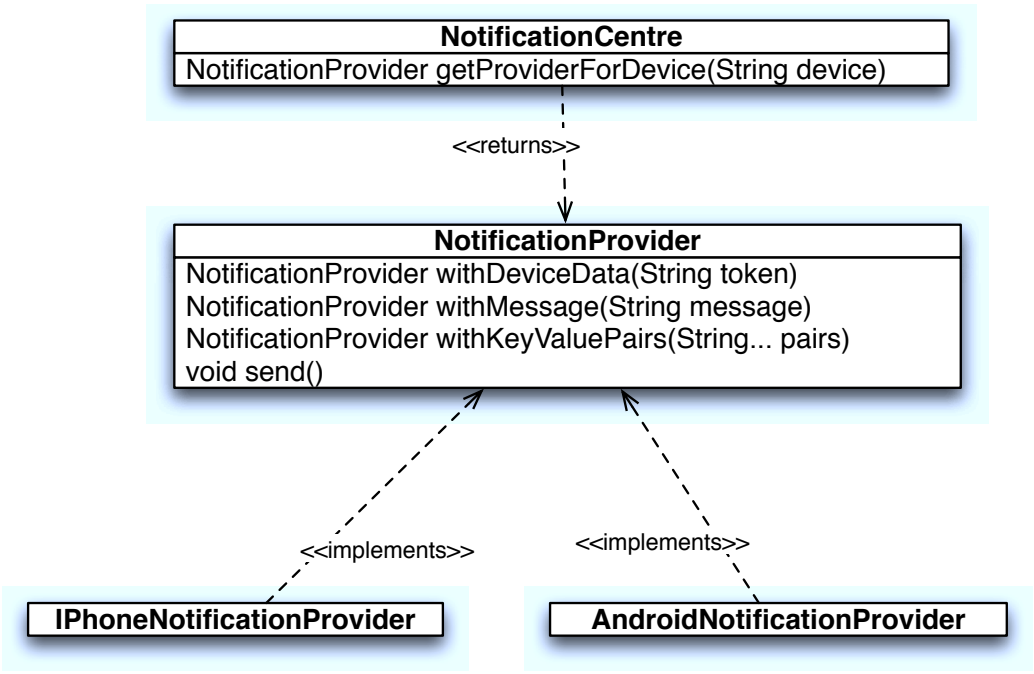

Figure 4.6: Messaging class diagram

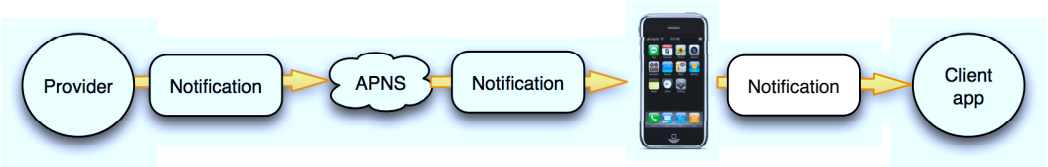

Figure 4.7: Message path using Apple Push Notification Service (APNS)

amount of time. Some of the tasks might require input from different users which can happen at unknown time. Thus maintaining TCP connection for getting feedback from MCM did not seem reasonable.

Other possibility would have been to use polling. The application would repeatedly request server for updates. This adds extra load for both the MCM servers and client application. Last of which is especially undesired because it wastes battery when done frequently. When doing it sparsely feeling of real-time response would be lost.

Apple mobile devices from iOS version 3.0 (June, 2009) have the possibility to use Apple Push Notification Service(APNS). It is an operating system feature where it will request notification info from apple servers. When there is a notification available for application running on particular device, user can choose whether he allows the application to handle it.

MCM takes advantage of APNS by sending out short notification consisting of information that the client application can then use to take further actions. These could be

requesting status of the task, asking results of the task being run or to provide additional input.

4.4.1 APNS prerequisites

Sending push notifications requires certificate to connect to APNS and device token to identify the receiving device.

Certificate is needed to establish secure communication channel with APNS server. Apple provides two binary interfaces for sending notifications: development interface at gateway.sandbox.push.apple.com and production interface gateway.push.apple.com. Both require separate certificates. Certificates necessary for establishing TLS (or SSL) connection are provisioned through the iOS Developer portal.

Device token is needed to uniquely address the receiving device. When application is started it must register at APNS to receive push notifications. After successful registration, callback will be made to application's application:didRegisterForRemoteNotificationsWithDeviceToken: method. At this point application knows it's token and must register it at provider (MCM web service).

To avoid sending messages to devices that do not have apple provides feedback service that gives information about devices that were sent a message but did not have the targeted application installed. In MCM context this might happen when user removes application he used to start task before the task completes.

4.4.2 APNS features

Values for alert, badge and sound keys in the notification payload JSON will affect how the message appears to user upon receiving.

Alert can be either a string or a dictionary for customized alert box.

Sound is name of an audio packaged in application's bundle. If the file is longer than 30 seconds or does not exist, defaul sound will be played.

Badge is a small red icon displayed at the top right of application's icon.

4.4.3 APNS constraints

There are certain limitations and aspects that developer must adhere to when sending notifications using APNS. Message payload must not be longer than 256 bytes and must not be null-terminated. APNS will close connection if these rules are violated.

Apple documentation states that APNS providers should determine and refrain from sending notifications to devices that do not have targeted application installed. Information about those devices can be acquired from Feedback Service. In MCM context this is rarely the case because users initiate job requests from application(s) that use MCM thus they very likely have the app installed at the time response notification is being sent.

Another consideration to be taken into account is that connection should be retained rather than re-initiated between sending multiple notifications. Making large number of connections in short period of time can lead Apple to identify this as DoS attack and block the provider's IP.

5 Developed prototype applications using MCM

Three proof-of-concept application were developed by author and fellow research group members. Goals of these were to test the features of MCM and measure it's performance.

5.1 Researcher's diary [iOS,Android][9]

Researcher's diary is an application that allows it's user to see upcoming conferences on a map, review materials related to particular conference and to upload new materials.

Application uses Google Maps for showing user's location on a map. Materials related to a conference are stored on Amazon S3. Amazon SimpleDB is used to keep track of different users, their conferences and materials. Materials among attendees of one conference are shared.

5.2 MCM Task Manager [iOS][8]

This application allows user to invoke different tasks on MCM. Tasks are presented as a list. User is provided with option to choose a file or enter additional data if the task to be started requires it. When a message regarding to requested task is received, application proceeds to check for results. Separate screen is used for displaying list of results.

MCM Task Manager application also exchanges and tracks timing information about the actions. Timing data is kept on MCM and can be requested when needed. Application has separate screen for showing this information.

5.3 CloudImageProcessing [iOS][7, 10]

CloudImageProcessing goal is to allow user to upload a video files and request processing of those uploaded. Processing makes use of concept similar to MapReduce patented by Google. Processing takes a video file and extracts each frame as a picture. Each picture is then searched for faces and specially eyes when found the eyes are covered with black box. After all frames have been processed, images are joined back to video file. After completing processing, user is notified. He then can request viewing of the new video.

Application is made up from three parts: cloud, web service and client application.

5.3.1 Server architecture

Server code was developed using Java Servlet API. Building and deploying was handled by Maven.

Server side that provides the services to client application currently consists of 4 servlets: UploadResource, ListResources, ProcessResource and ResourceThumbnails. Their functions and parameters are as follows:

UploadImage

INPUT multipart/form-data HTTP POST request with one file as it's data

OUTPUT Nothing

Description Is used for uploading video file to server only. This is a synchronous task meaning that no additional information regarding to a current request will be sent after servlet returns.

ListResources

INPUT Nothing

OUTPUT JSON formatted array of VideoResource objects that can be processed.

Description Should be requested each time when list of resources needs to be updated. This request currently takes about 20 seconds when used with Tomcat server instance running in Amazon EC2 and bucket containing about 500 items. It is realized synchronously on server and client application should make this request in background and use device specific tools for making this request in background (threading for example).

ProcessResource

INPUT One VideoResource object in JSON representation

OUTPUT Processed file's VideoResource JSON representation

Description This servlet is used to start the processing of a video file. The file must already be present in a location that server knows. Easiest way for it is to aquire the VideoResource object from the output of ListResources servlet. This servlet's behaviour is asynchronous. Since the processing of a video file can taks quite a long time and client can not be expected to be available (e.g. because of connection loss or closing of the application while waiting). APNS (Apple Push Notification Service) is used to inform client about completion of a processing task. Notification, sent to client, contains JSON representation of the resource that was the result of processing.

ResourceThumbnails

INPUT VideoResource object's JSON representation

OUTPUT List of URLs in JSON format

Description URLs in response link to the locations of thumbnail images for specified resource.

5.3.2 iPhone application

iPhone application developed serves the purpose of demonstrating the functions of server. After the application is launched, it makes a request for list of resources (ListResources servlet) to server in background. Initially empty table is being shown. When the resources list arrives it will display them as shown in Figure 5.1 on page 21. After user selects a video from the list it's details would be shown as seen in Figure 5.2 on page 22

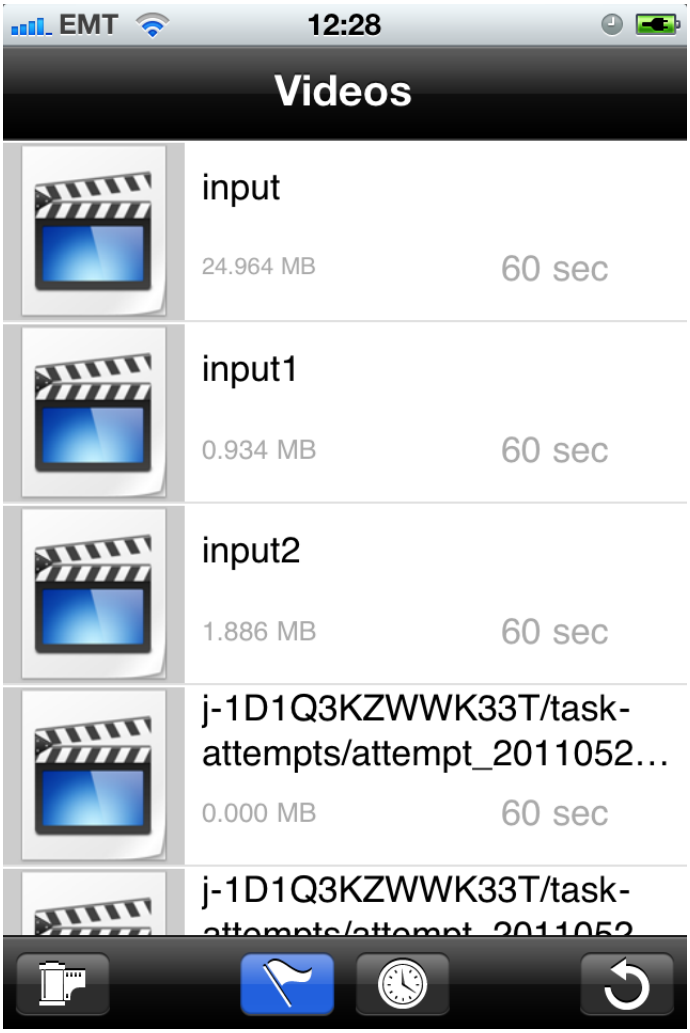

Figure 5.1: List of video resources available

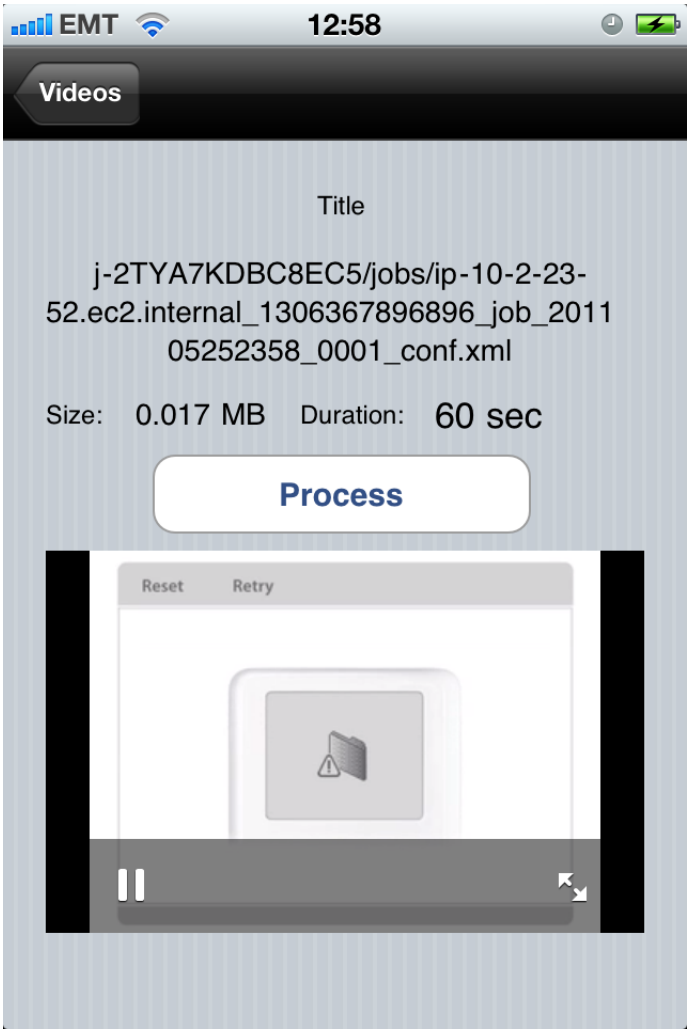

Figure 5.2: Resource details screen

6 Performance analysis

To get better overview of how the synchronous and asynchronous models behave, times for different operations they performed were measured. Asynchronous behaviour in this case means that HTTP connection would be kept up until the processing of the client's request is completed.

6.1 Performance model of the MCM

In a regular mobile cloud service invocation cycle, as shown in 6.1 on the next page, the total time of invocation includes the transmission delays and the time to process the requested task at the cloud. So the time taken to handle a cloud service request can be calculated as:

$$T_{csr} \approxeq T_{tr} + T_c \tag{6.1}$$

Where T_{tr} is the transmission time taken across the radio link for the cloud service request. The value includes the time taken to transmit the request to the cloud and the time taken to send the response back to the mobile. Apart from these values, several parameters also affect the transmission delays like the TCP packet loss, TCP acknowledgements, TCP congestion etc. So a true estimate of the transmission delays is not always possible. Alternatively, one can take the values several times and can consider the mean values for the analysis. T_c is the time taken to process the actual service at the cloud. $\approxeq$ is considered in the equations as there are also other timestamps involved, like the client processing at the mobile phone, as shown in 6.2 on the following page. However, these values will be quite small and cannot be calculated exactly.

When the MCM is introduced to the invocation cycle, the invocation cycle transforms to the one shown in 6.2 on the next page. Here, the total mobile cloud service invocation time, T_{mcm} is:

$$T_{mcm} \approxeq T_{tr} + T_{te} + T_c \tag{6.2}$$

T_{tr} becomes transmission time across the radio link for the invocation between the mobile phone and the MCM. T_m is the time taken to process the request at the middleware. T_{te} is the transmission time across the Internet/Ethernet for the invocation between the middleware and the cloud. T_c stays the same as in 6.1.

From 6.2 we can observe that extra delays are included to the invocation cycle with the middleware in place. However, apart from the main benefit of interoperability that is brought in by MCM, several observation also make the middleware a better solution even in terms of the performance. Consider the case of synchronizing the data across

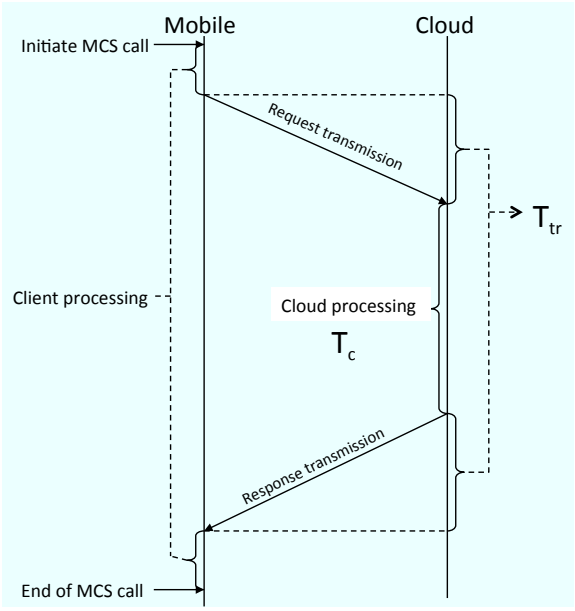

Figure 6.1: Mobile cloud service invocation cycle with regular case

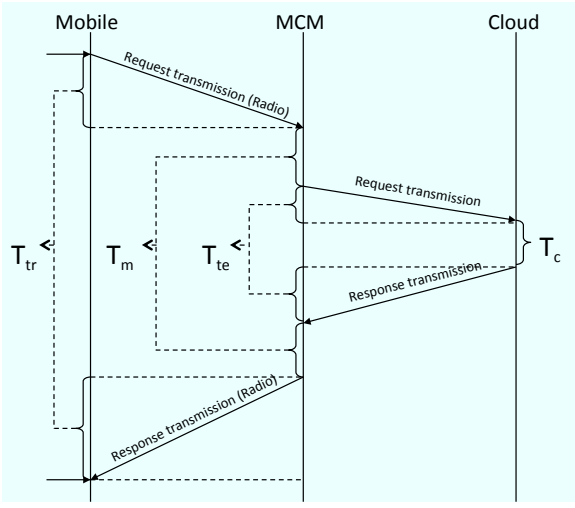

Figure 6.2: Mobile cloud service invocation cycle using MCM

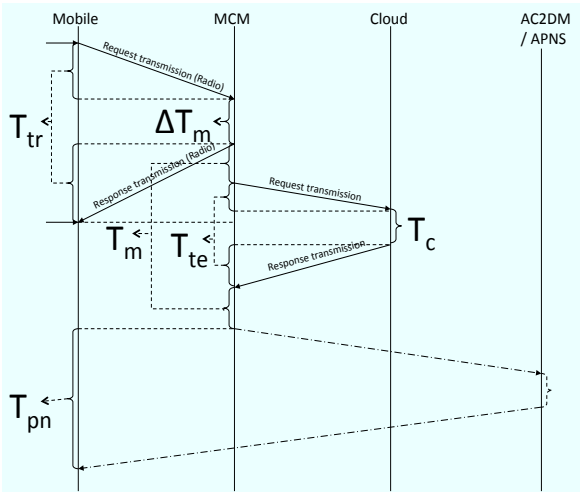

Figure 6.3: Mobile cloud service invocation cycle when using MCM and asynchronous messaging

multiple clouds or having to perform parts of the service across multiple clouds; In this case the regular mobile cloud service invocation cycle time becomes:

$$T_{mci} \approxeq \sum_{i=1}^{n} (T_{tr_i} + T_{c_i}) \tag{6.3}$$

Where as in the case when the MCM is involved int the invocation cycle time becomes:

$$T_{mcm} \approxeq T_{tr} + T_m + \sum_{i=1}^{n} (T_{te_i} + T_{c_i}) \tag{6.4}$$

However, the transmission delays across the radio link are far greater than the transmission delays across the Internet due to bandwidth being smaller in mobile networks. Assuming $T_{te} \lll T_{tr}$, the middleware solution quickly outperforms the regular solution as the value of i increases in 6.3 and 6.4.[14]

6.2 Measuring times

For the performance analysis, timestamps for different operations had to be measured. If an operation was started and ended on one system then start time could be subtracted from end time and the result would be the duration. When an operation was started on server and ended in client or vice versa, those timestamps would not be directly comparable. The reason is that two clocks could be out of sync. To compare those times difference between two clocks has to be known. Time difference was calculated using method consisting of following steps.

1. It was assumed it takes the same amount of time for the request to travel from client to server as for the response from server back to client.

2. Following timestamps were measured
   a) T_{cir} in client when it initiated request
   b) T_{srr} in server when it received request. Server would respond immediately. No processing would be done. Response would contain time stamp T_{srr}
   c) T_{crr} in client when response from server was received.
3. Assuming 1. is correct (with small possible mistake) and clocks are in sync T_{srr} should happen exactly at $T_{cir} + \frac{T_{crr}+T_{cir}}{2}$. The difference between $T_{cir} + \frac{T_{crr}+T_{cir}}{2}$ and T_{crr} would be the amount of time that clocks are out of sync.
4. t_{diff} time difference between device and server clocks can thus be calculated: $t_{diff} = T_{cir} + \frac{T_{crr}+T_{cir}}{2} - T_{srr}$. If this value is positive the server clock is in front, if negative, server clock is behind.

t_{diff} was calculated multiple times and average was taken. After the sync difference was known timestamps measured in client and server could be compared.

6.3 Performance analysis and results

To study behaviour in time CloudImageProcessing application, also described in 5.3 on page 18 was used. This application is good candidate for being studied because it uses cloud resources via MCM multiple ways. Users' video files are being stored in the cloud (at the time of writing Amazon S3 was chosen) and also processing of selected video file is done in the cloud. Namely MapReduce algorithm provided by Hadoop framework.

Hadoop is a framework that provides support for the analysis of data-intensive distributed applications (thousands of nodes) within the cloud. The algorithm applies map/reduce for matching analysis of a vast amount of data. Hadoop supports parallel computation and distributed file system. The MapReduce algorithm consists of two basic steps, the Map and the Reduce functions. The Map function takes one set of key value pairs and maps them to intermediate key pairs. The intermediate key pairs are sorted and grouped together by the framework and passed to the reduce function. The Reduce function takes the intermediate key pairs and produces the output.

To measure the times, some changes had to be made to the application. Model class AsynchronousTestTimes shown on 6.4 on the next page was defined. It's member variables were named according to where they would be used. All values would initially be -1. When a measurement is being done, relevant member variable would have be initialized. Timestamps in milliseconds from 1. January 1970 were chosen.

Some of the times would be measured on server and some on the client device. Results for one test would somehow have to be collected into one instance of AsyncTestTimes. For that TestTimesServlet was developed. It takes 3 HTTP POST parameters: testID, testType and clientTestTimes. testID is unique string used to identify and associate an instance of AsyncTestTimes class with the resource that was processed. They both would have the same ID. testType could be either "AsyncTests" or "SyncTests". Currently only

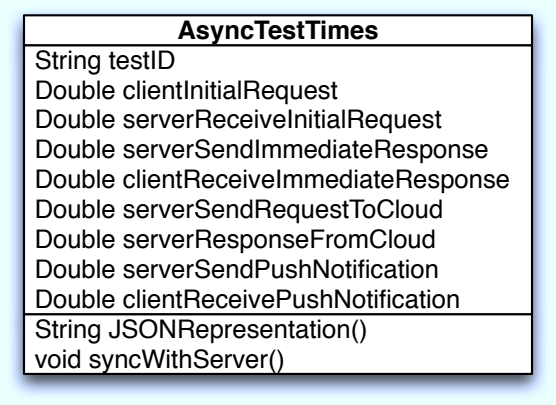

Figure 6.4: Asynchronous test times class diagram

"AsyncTests" is relevant and is used. clientTimes is JSON formatted string of the times object being sent over.

When client sent new times to server using TestTimesServlet it would be enough for it to only initialize testID and those fields of the AsyncTestTimes object that it wishes to update. Other fields would have to be equal to less than 0 to be ignored.

If TestTimesServlet was called without parameters, it outputs all test times in server using CSV (Comma Separated Values) format.

Since the clocks can not be assumed to be in sync on server and device, synchronization mechanism described in 6.2 on page 25 was also developed. Each time synchronization would be requested, 10 passes would be done and average of those and the ones from previous times would be calculated. The result of this calculation could be used "as-is" when calculations are done not long after. That was the case here. Once the sync difference was known, it was defined as a constant in TestTimesServlet. When TestTimesServlet is called

6.3.1 Experiment description

Timestamps were measured for processing a video with size 10.5 MB and duration 10 minutes. Device used for initiating the processing was iPhone 4 with iOS 4.3.3. It had 512 MB of eDRAM and Apple A4 (ARM Cortex-A8 based) processor. Device was connected to internet using WiFi network.

Two buttons  ,  were added. Whole user interface can be seen 5.1 on page 21. First one runs the sync difference measurement process. Second one requests processing of the video from MCM. When second button is pressed test is being run an times calculated. Test includes following steps:

1. Client starts the processing task by pressing middle right button on screen. runTestsPressed: method will be executed.

2. iOS application creates new MCAsyncTestTimes instance and sets it's clientInitialRequest property with current timestamp. MCVideoResource with the same ID as MCAsyncTestTimes instance is created.

3. Request to MCM is being made. MCVideoResource object JSON description will be passed as a HTTP POST parameter.

4. MCM receives the request. It will create it's own AsyncTestTimes instance with the ID from VideoResource object passed in as JSON. MCM stores the AsyncTestTimes (that now has same ID as VideoResource) using TestTimesManager

5. MCM creates instance of ProcessResourceTask and passes the described VideoResource instance to it. Created ProcessResourceTask instance is runnable and will be run in separate thread.

6. MCM records current time stamp for serverSendImmediateResponse in AsyncTestTimes instance.

7. Client receives response for a request initiated in 3. It will record current time stamp for clientReceiveImmediateResponse. Client will store the MCAsyncTestTimes instance using MCTestTimesManager.

8. When ProcessResourceTask's run method is invoked it will ask TestTimesManager for instance of AsyncTestTimes with the same ID as it's VideoResource. It will record the time stamp in serverSendRequestToCloud before making request to cloud.

9. ProcessRequestTask will process video.

10. When processing video is complete it will record current time stamp for AsyncTestTimes member variable serverResponseFromCloud instance it got in 7.

11. ProcessRequestTask will record time stamp for serverSendPushNotification and send out push notification

12. Client receives push notification. It has resourceID within it. It will then ask for MCAsyncTestTimes instance from MCTestTimesManager and set value for clientReceivePushNotification.

13. Client synchronizes it's MCAsyncTestTimes instance with server using former's method syncWithServer.

14. TestTimesManager in MCM now has all times for the AsyncTestTimes instance.

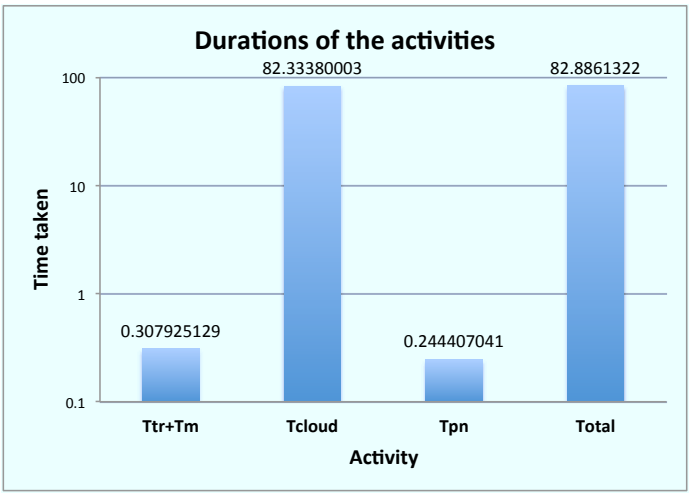

Figure 6.5: Results of the tests

6.3.2 Results

Processing of the video were requested 5 times and timestamps were measured as described in previous section. After having the timestamps, durations according to MCM asynchronous invocation cycle with push notifications, were calculated. The model is shown 6.3 on page 25.

From the results chart 6.5 it can be seen that the time it takes to process the request in the cloud is longest by a large margin. It should be noted that the vertical axis is with logarithmic scale. By using MCM with asynchronous messaging, device user can be left with an impression that he is in total control. Even when Tpn (duration for the push notification to arrive at client) would be significantly larger, user would not notice it because he would attribute it to the performing of the task. Although similar responsiveness can be achieved by running the longer lasting tasks in background threads synchronously it would not be a good choice. The reason behind it is that during the processing, user HTTP connection could be interrupted for various reasons (e.g. poor WiFi coverage, empty battery, closing of the application). Asynchronous messaging used in MCM allows developer to separate performing the task from notifying owner completely.

7 Conclusion

This thesis explored the idea of invoking resource intensive tasks from the mobile phone and executing them in the cloud. The number of cloud service providers and different tasks they allow to provide is continuing to grow. Many of the cloud service providers also provide API-s for mobile devices. While it is sometimes sufficient to only consume the cloud services directly from devices, this approach has some serious weaknesses. They manifest themselves when interoperability (e.g. in sense of moving data between clouds) is required or developers would like to change the cloud resource provider. When provider would change their API it could brake the application.

Solution to these problems is to add an extra layer of abstraction in a form of web service. Through that application's functionality would be provided. Client application would invoke services from web service and web service would communicate with cloud. To use this kind of architecture, asynchronous behaviour of server is required. In this thesis web service and client applications with this kind of asynchronous behaviour were developed. Messaging services provided by Apple were used to notify client device about status changes.

From the performance analysis it can be seen that using asynchronous messaging enables user to have better experience with the application. The time it takes to do the actual task is longer by a large margin, than the additional operations needed to implement the asynchronous messaging. Additional time it takes the asynchronous message to travel is hardly every noticable. For the developer, using asynchronous messaging and MCM means that he can contact device and not worry about things like interruption of a HTTP connection during the processing of a task.

8 Future research

Most of the code written for this thesis was with the goal of proving or trying out specific concept or idea. To make the results more usable for commercial applications and other developers, further development would be needed. Important part of it would be to thoroughly test it and provide base classes for common use scenarios.

Security issues is aspect that should be considered before this architecture and implementation could be used commercially. Currently all data is being sent over HTTP in plain text. Using HTTPS for exchanging data would get rid of some problems. Current system does not implement any authentication mechanism. Currently only different devices are distinguished. Having many users and possibly even multiple users on the same device would require solving these issues.

Another security issue is keeping the credentials for accessing different cloud and other resources safe on server. One solution to that would be to keep them in a private folder, separate from the web application jar file. When deploying new application's war remotely to cloud instances updated files would have to copied also.

The implementation of MCM right now does not have any persistent storage integrated. This means that all server state is being kept in memory. Singleton classes are used to manage stores of different objects. This suffices for the purpose of the thesis but in future keeping users' data in some kind of persistent database would be required. It is quite probable that the database would not reside on the same machine as the web server. Security issues related to that would have to be addressed.

To allow other developers and companies use the ideas presented here set of different generic example applications would have to be created. Those applications would demonstrate different aspects of MCM and would be available for all popular mobile platforms (e.g. iOS, Android, Windows Phone7).

9 Sisukokkuvõte

Pilvearvutuse ja mobiilse pilvearvutuse valdkonnad arenevad kiiresti ja paistavad paljulubavate tehnoloogiatena tulevikuks. Mobiilsete seadmete arvu kasvades muutuvad neile loodud rakendused keerulisemaks ja "targemaks". Et töödelda varasemast suuremaid andmehulki on arendajad pöördunud pilveteenuste poole. Pilveteenuste kasutamine muutub mobiilseteks kui seda kasutatakse mobiilirakenduse tarvis. Mobiilsed seadmed saavad kasutada pilveteenuste paindlikkust selleks, et kaugligipääsu abil täita töömahukaid ülesandeid, jätkates samas pehmes reaalajas toimimist. Et hõlbustada pilveteenuste kasutamist mobiilsetest seadmetest on Satish Srirama juhitavas uurimisrühmas tegeletud MCM (Mobile Cloud Middleware) ehk eesti keeles mobiilse pilve vahevara loomisega. MCM eesmärgiks on lihtsustada erinevate pilveteenuste vahelist andmete liigutamist ja pakkuda neile ühtset liidest. Kui mobiilirakendus sõltub suurte andmehulkade ja rohke protsessoriaja kasutamist nõudva ülesande täitmisel pilveteenustest, siis ei tohiks ülesande täitmise ajaks telefon "lukustuda". Lahenduseks sellele probleemile pakub käesolev töö välja asünkroonse teadete saatmise lahenduse MCM-s.

Bibliography

[1] Abi research - technology market research [online]. Available from: http://www.abiresearch.com/. 7

[2] Amazon web services [online]. Available from: http://aws.amazon.com/. 7

[3] Cloud computing, managed hosting, dedicated server hosting by rackspace. Available from: http://www.rackspace.com/. 7

[4] Mobile cloud applications. Technical report, ABI Research, 2010. Available from: http://www.abiresearch.com/research/1003385. 7

[5] Mahmood Ali. Java apple push notification service provider [online]. Available from: https://github.com/notnoop/java-apns. 14

[6] Huber Flores. Mobile cloud middleware. Master's thesis, University of Tartu, 2011. 12

[7] Madis Nõmme. Cloudimageprocessing ios application github repository [online]. Available from: https://github.com/ut-mobile-cloud/cloudimageprocessing-ios. 2, 18

[8] Madis Nõmme. Mcm task manager github repository [online]. Available from: https://github.com/ut-mobile-cloud/MCMMessaging-ios. 2, 18

[9] Madis Nõmme. Researcher's diary [online]. Available from: https://github.com/ut-mobile-cloud/LinkedPoints-ios. 2, 18

[10] Madis Nõmme and Carlos Paniagua. Cloudimageprocessing server github repository [online]. Available from: https://github.com/ut-mobile-cloud/cloudimageprocessing-server. 2, 18

[11] Dave Rosenberg. Why mobile applications need cloud services [online]. Available from: http://news.cnet.com/8301-13846_3-10300564-62.html. 7

[12] S. N. Srirama, V. Shor, E. Vainikko, and M. Jarke. Supporting mobile web service provisioning with cloud computing. *International Journal On Advances in Internet Technology*, 3:261–273, 2010. 4, 8

[13] Satish Narayana Srirama, Huber Flores, and Carlos Paniagua. Handling process intensive hybrid cloud services from mobiles. 2011. 6

[14] Satish Narayana Srirama, Carlos Paniagua, and Huber Flores. Croudstag: Social group formation with facial recognition and mobile cloud services. 2011. 25